Comp422
Project 1

Boxiong Tan 300300835

Introduction:

This project contains three parts. The first is Image preprocessing and basic operators. In this part, we implement some basic convolution algorithms in order to achieve image edge detection, noise cancellation and image enhancement. In the second part, firstly, we implement two ideas in order to filter out the larger galaxies. Secondly, we implement a naive bayes classifier that could classify human face and non-face. Furthermore, based on output result, we use TPF/FPF to measure the performance of the classifier as well as the feature selection. In the third question, we need to make use of the extracted features of hand-written digits from raw resource files. Then we use C4.5/J48 algorithm to classify them. After we get the result, we give evaluation of the results and make a comparison between results.

The structure of the content are as follow, in question 1, we would first show the result, then explain methods and how do we implement it, the last step is evaluation.
In question 2, we would first explain the methods, then shows the procedure. The next step is explaining the results. Finally we give evaluation and discussion. In question 3, we would explain how to make use of the provided features and then gives evaluation of the results and discussion.

Tools and Environment:

Operating System: Unix/Linux
Programming Language: Python, Shell, R

Question 1:

1.1 Edge detection:

1.1.1 Results:

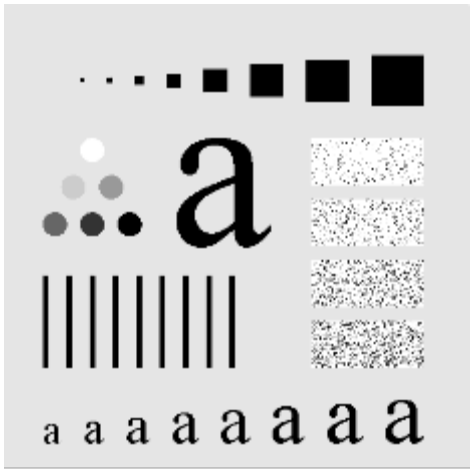


Figure 1

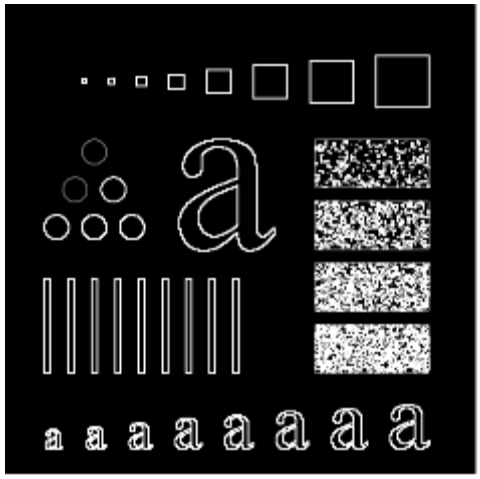


Figure 2

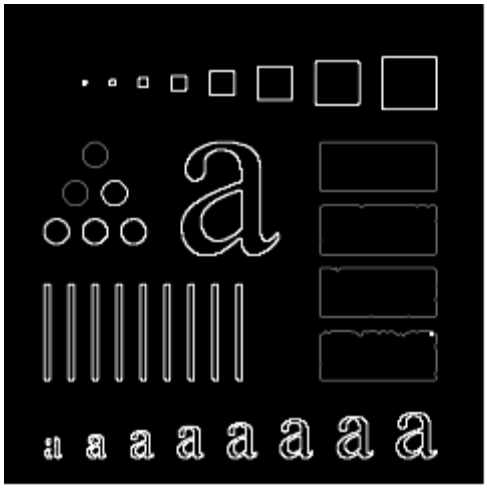


Figure 3

Figure 1 is the original picture.
Figure 2 shows the result of implementing sobel operator convolution.
Figure 3 shows the result of implementing median filter and sobel operator convolution.

1.1.2 Explain Methods:

Figure 1 —Sobel operator Convolution→ Figure 2

Figure 1 —Median Filtering→ temporary Figure —Sobel operator Convolution→ Figure 3

Sobel operator:

In image processing, a **kernel**, **convolution matrix**, or **mask** is a small matrix useful for blurring, sharpening, embossing, edge-detection, and more. This is accomplished by means of convolution between a kernel and an image.

The Sobel edge detection uses two 3 * 3 kernels which convolve image.
The computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

A is the 3 * 3 pixel squares.
At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

After we get the G value, we set the centre pixel of A to G. As we scan the image, we do the convolution over the image.

1.1.3 Implementation:
Our source code sobel.py store in Question_1/src/ is an implementation of sobel method using Python.

The program would first read the original image, store in the memory represented as a 2-d array. Then do the convolution with sobel operator row by row. Then we write a new file with convolved array. The result will be generated in pics/result/ directory with the name sobel_test.png

1.1.4 Discussion:
As we can see, in Figure 2, the noises could not be handled well using sobel's method.
In order to deal with noise problem, we implement median filter on original image first and then use sobel method to get the edge detection. Figure 3 shows the result, it is much better than solely using sobel method.

Because the Median filtering is particularly good at remove salt and pepper noises. The median filter would be explicitly explained in Question 1.2.

1.2 Noise Cancellation

1.2.1 Results:

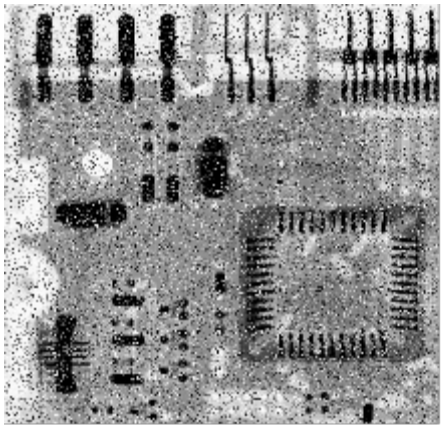


Figure 4

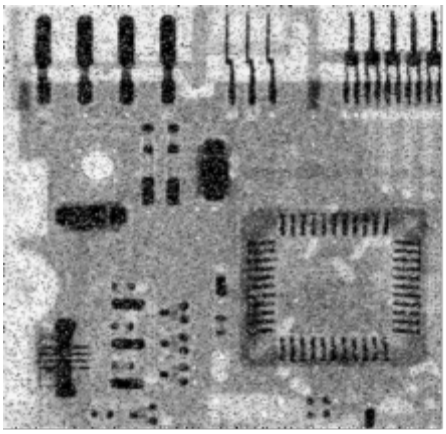


Figure 5

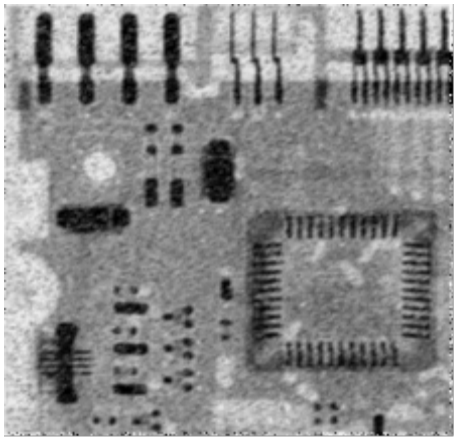


Figure 6

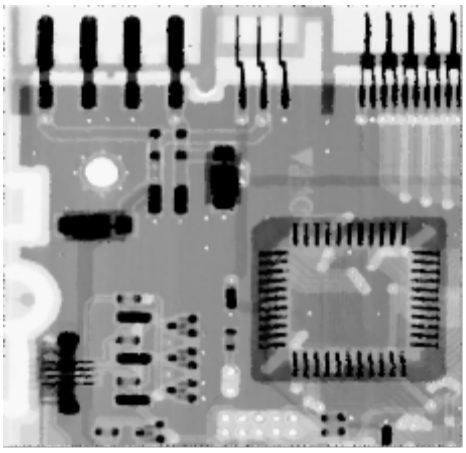


Figure 7

Figure 4 is the original picture.
Figure 5 shows the result of using 3 * 3 kernel mean filter.
Figure 6 shows the result of using 5 * 5 kernel mean filter.
Figure 7 shows the result of using median filter.

Mean 3 * 3 kernel

Figure 4 ⟶ Figure 5

Mean 5 * 5 kernel

Figure 4 ⟶ Figure 6

Median 3 * 3 kernel

Figure 4 ⟶ Figure 7

1.2.2 Explain Methods:

Mean filtering:

The idea of mean filtering is simply to replace each pixel value in an image with the mean (‘average’) value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown in Figure 5, although larger kernels (*e.g.* 5×5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel.)

Median filtering:

The median filter replaces a target pixel's value with the median value of the neighbouring pixels(*e.g.* 3×3 squares).

1.2.3 Implementation:

Mean filter:

Our source code mean_3.py store in Question_1/src/.

In mean filtering, we employ a mean kernel:

```
kernel = {
        1/9, 1/9, 1/9
        1/9, 1/9, 1/9
        1/9, 1/9, 1/9
        }
```

By using this kernel,we do convolution pixel by pixel. And lastly write the convolved array into a file called mean_3_ckt.png stored in ./pics/result/.

For the larger kernel, we employ a larger mean kernel(5 * 5 square):

```
 kernel = {
        1/25, 1/25, 1/25, 1/25, 1/25,
        1/25, 1/25, 1/25, 1/25, 1/25,
        1/25, 1/25, 1/25, 1/25, 1/25,
        1/25, 1/25, 1/25, 1/25, 1/25,
        1/25, 1/25, 1/25, 1/25, 1/25,
        }
```

The generated image named mean_5_ckt.png stored in ./pics/result/.

Median filtering:
In this approach, first we need to extract a 3 * 3 pixel square from original image. Then we sort this matrix and find out the median value. Then, we set the median value to the centre pixel of the extracted matrix. The last step is to write the pixel into the image.
The result is named median_ckt.png store in ./pics/result/.

1.2.4 Discussion:
For the mean method, the larger operator matrix we use, the smoother the result will be. In contrast, we can see in the Figure 6, We employ a 5 * 5 square matrix. The result is much smoother than Figure 5.

Figure 7 clearly shows that this approach provide much better result than mean filtering. Because the salt and pepper noises are mostly 0(black) or 255(white) in grayscale. They will surely be get rid of if we use median filtering.

1.3 Image Enhancement
1.3.1 Results:

Figure 8

Figure 9

Figure 8 is the original image.

Figure 9 is the sharpened image.

1.4.2 Explain Methods:

The laplacian kernel emphasize the edges on the original image.

1.4.3 How to implement:

In the question, We employ laplacian kernel and do the convolution.

```
kernel = {
            0, -1, 0
            -1, 4, -1
            0, -1, 0
    }
```

The result is named sharp_moon.png store in ./pics/result/.

Question 2.1 Mining Space Images

2.1.1 Results:

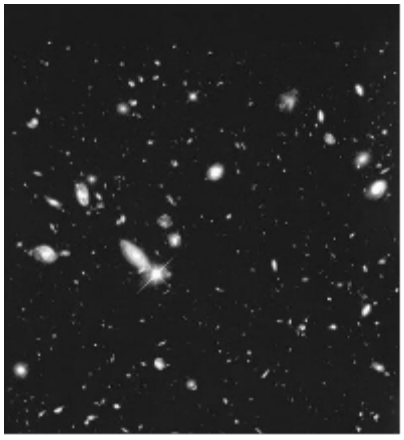


Figure 10

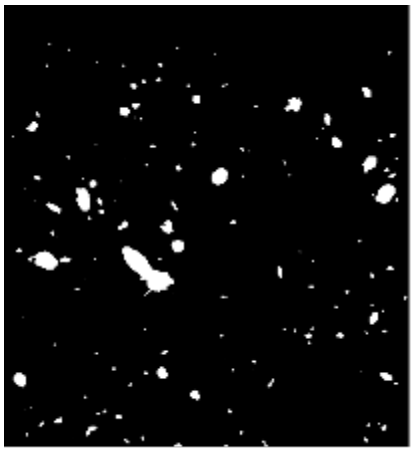


Figure 11

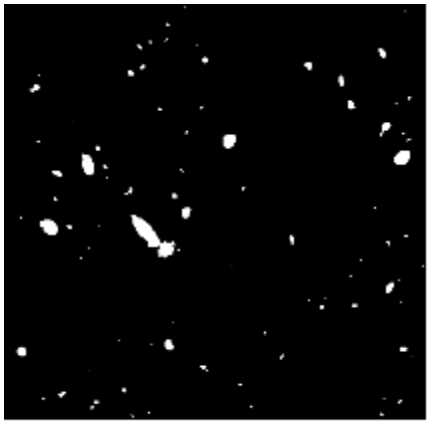
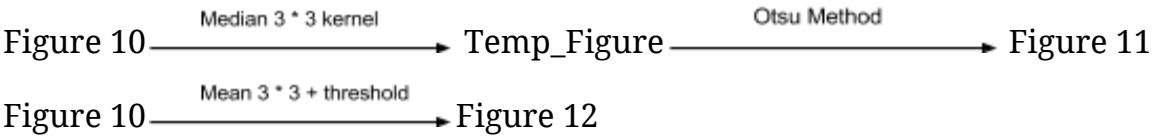


Figure 12

Figure 10 is the original image.

Figure 11 shows the result of using median filter and otsu's method.
Figure 12 shows the result of using mean filter and threshold.



2.1.2 Explain Methods:
In this question, we are suppose to mark those large galaxies and turn the image to a black-and-white image. We design two ways to achieve that.

Median Filter and Otsu's Method
In order to get rid of those "tiny galaxy" on the image, intuitively speaking, the tiny galaxies are similar to salt and pepper noise. So median filter is a good choice to preprocess this image. Then we employ Otsu's method to automatically find the threshold that differentiate the background and the objects.

The Otsu's method is used to automatically perform clustering-based image thresholding[1], or, the reduction of a gray level image to a binary image. It assumes that the image contains two classes of pixels following bimodal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is maximizing.

In Otsu's method it exhaustively searches for the threshold that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of variances of the two classes:

Weights W_1 and W_0 are the probabilities of the two classes separated by a threshold t.
N_1 is the number of pixel of the foreground object.
N_2 is the number of pixel of the background.
M is the width and N is the height of the image.
M * N is the number of whole pixels in image.
$\boldsymbol{\mu}_i$ is the mean value of grayscale in each class(foreground image and background).
$\boldsymbol{\mu}$ is the mean value of grayscale for the whole image.
g is the intra-class variance.

$$\omega_0 = \frac{N_0}{M * N} \quad (1)$$

$$\omega_1 = \frac{N_1}{M * N} \quad (2)$$

$$N_0 + N_1 = M * N \quad (3)$$

$$\omega_0 + \omega_1 = 1 \quad (4)$$

$$\mu = \omega_0 * \mu_0 + \omega_1 * \mu_1 \quad (5)$$

$$g = \omega_0(\mu_0 - \mu)^2 + \omega_1(\mu_1 - \mu)^2 \quad (6)$$

Algorithm:

1. Count number of each gray scale value.
2. Calculate the probability of each gray scale.
3. Step through all possible thresholds t = 1 maximum = 255
    1. calculate ω_i and ω_i * $\boldsymbol{\mu}$i
    2. Calculate $\boldsymbol{\mu}$i
    3. Calculate g
    4. If the g is the maximum, then threshold = t
4. Return threshold

After we have the threshold value, we could use convolution to rewrite the image. Every pixel that greater than the threshold will be white and every pixel smaller than threshold will be black.

Mean filter with Threshold method

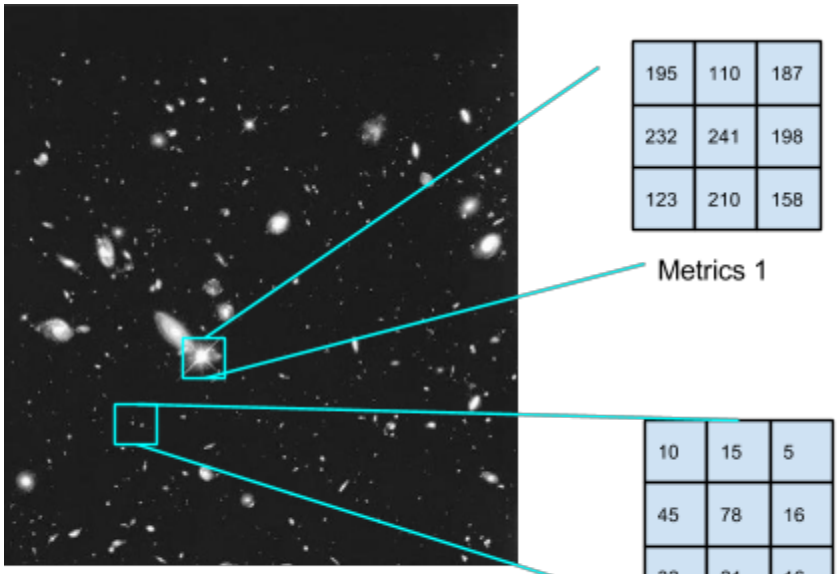


Figure 13

By observing the galaxy image, we notices that the **luminance** of the larger galaxies is different with the smaller galaxies.
As we can see in Figure 13: The pixel matrix 1 is extracted from a large galaxy. The pixel matrix 2 is extracted from a small galaxy.
In matrix 1, the average gray level = 183.7 much higher than matrix 2 = 26.4
Based on this observation, we pre-defined a threshold. As we extracted pixel matrix from image, we calculate the average value of the gray level. If the average value is greater than threshold, we set the central pixel white. Otherwise we set the pixel black.

2.1.3 Implementation:
Median filter and otsu's method
We first employ median filtering on the image. Because the tiny galaxies are more close to salt and pepper noise. In fact, if we repeatedly do the median filtering, the result will be better and better.

Second step, We employed Otsu's method to find a proper threshold to separate the background and the objects.

The result image is named large_galaxy_hubble.png

Mean filter and threshold
We pre-defined a threshold. As we extracted pixel matrix from image, we calculate the average value of it. If the average value is greater than threshold, we set the central pixel white. Otherwise we set the pixel black.

The result image is named threshold_galaxies.png

2.1.4 Discussion:
Although using median filter could effectively get rid of those tiny galaxies, we couldn't determine the size of the galaxy that needs to be retained. We could only repeatedly do median filtering until we are satisfied with the result.

In comparison with median filter and otsu's method, mean filter and threshold method could use threshold to determine the luminance of retained galaxies.

However, there are some disadvantages in mean filter and threshold method. Firstly, we define the threshold manually. The threshold could only be determined by observation and changing manually. In Figure 12, we use threshold = 150.

Secondly, although we retain the larger galaxies, the shape of galaxies are changed by the method. Usually the retained galaxies will be smaller than original image.

In conclusion, both methods could achieve the goal. However, both methods have disadvantages.

Question 2.2 Face Detection

In this part, we are provided with a collection of images that are the result of moving a window over a number of original images. The provided has been categorized into training set and test set. Each of training and test set contains face sets and non-face sets.

2.2.1 Explain Methods:

In the first step, we are suppose to extract some features from face training sets. Secondly, we calculate the mean value and variance of these features. The third step is to implement a naive bayes classifier, feed with the learned knowledge and then use it to classify the test face and non-face.After we gain the result, we would use TPF/FPF to evaluate the result.

2.2.1.1 Feature Selection:
The original training image is in grayscale, the first step is to change the image to black-and-white image using otsu's method as we mentioned in the Question 2.1. Then, as we can see in Figure 14. Based on common understanding of human face, we select eight features: left eye, forehead, right eye, nose bridge, left cheek and right cheek, nose and mouth.

We use the proportion of the black area of the square to measure each feature.

For example:
On the left corner is the left eye square, the proportion of the black area account for 80% of its square area. So we record this feature as 0.80.

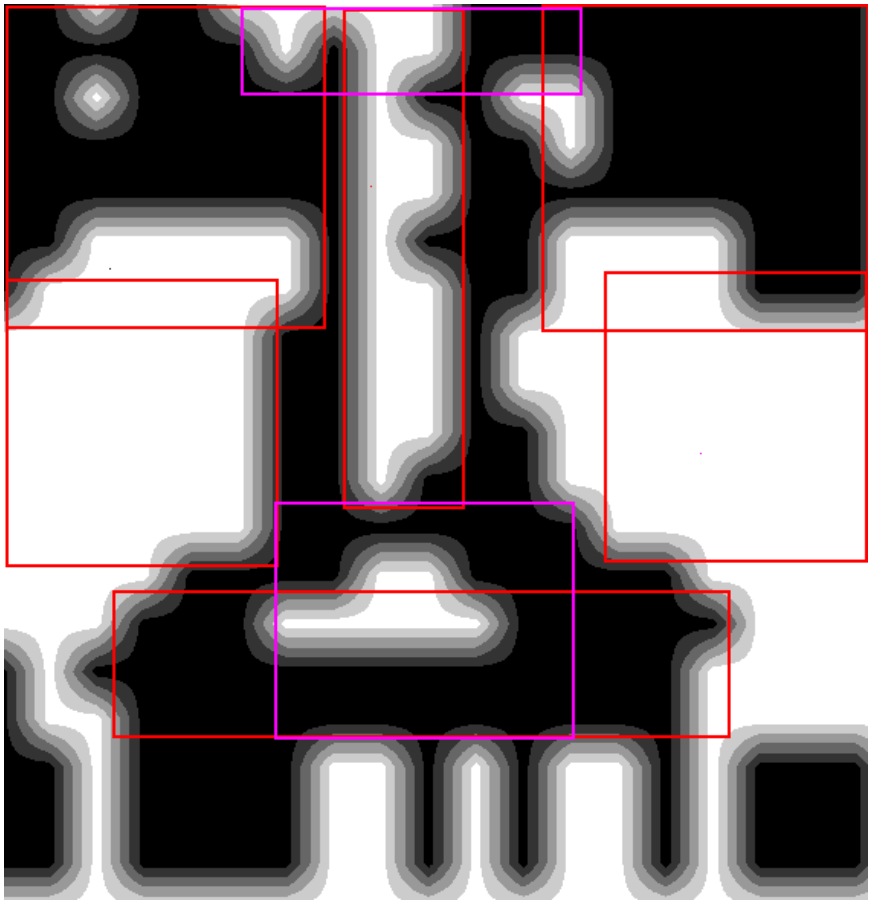


Figure 14

2.2.1.2 Feature Extraction:
As every image is a matrix of 19 * 19 pixels, we could manually define every feature as a square. Then we calculate the proportion of black area of the square.

For example:
On the left corner is the left eye square, it starts from row = 0 and column = 0 to row = 6 and column = 6.

2.2.1.3 Naive Bayes Classifier:

The structure of naive bayes classifier is shown in Figure 15:

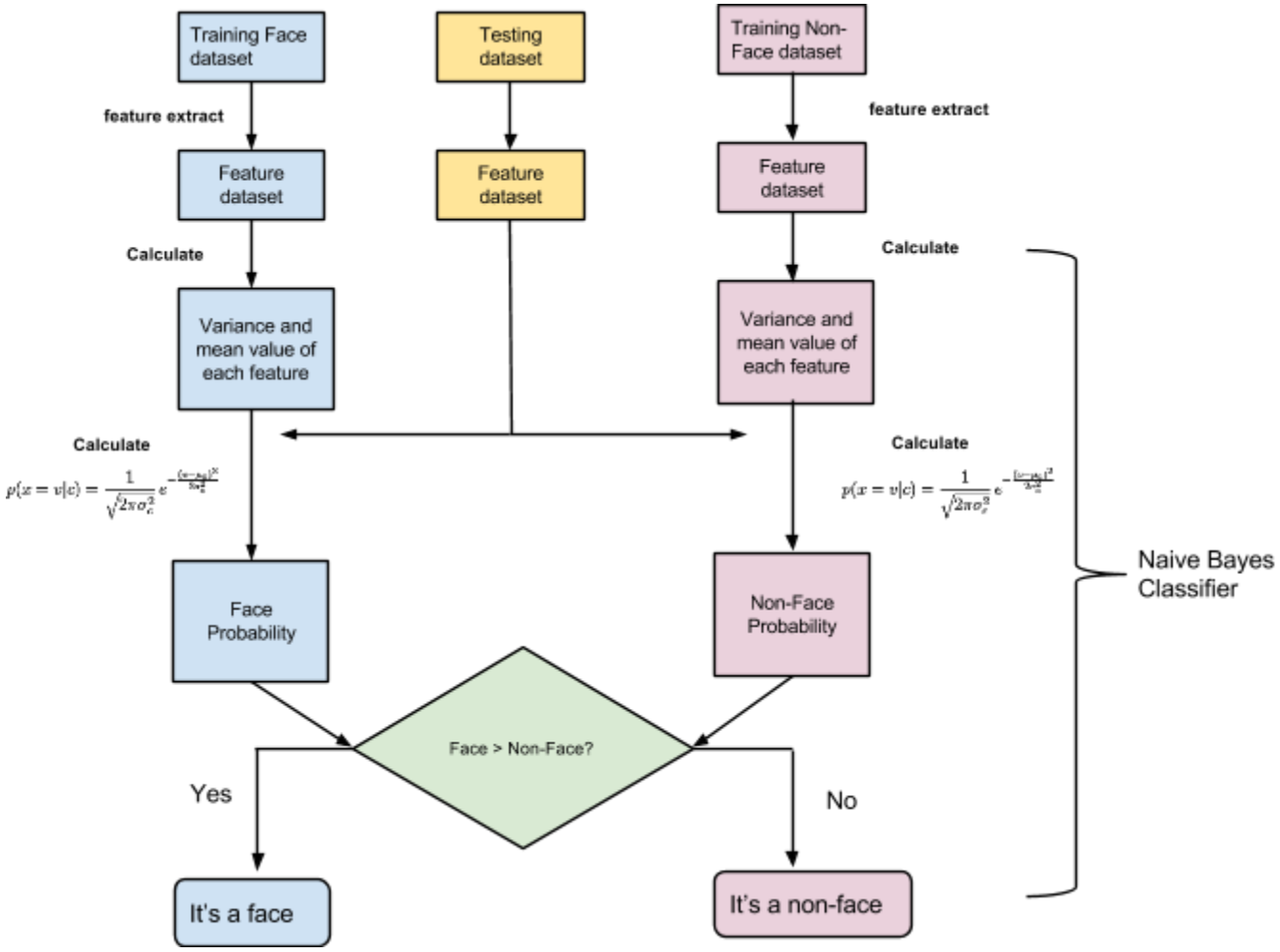


Figure 15

naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Using Bayes' theorem, this can be written

$$p(C|F_1,\dots,F_n) = \frac{p(C)\ p(F_1,\dots,F_n|C)}{p(F_1,\dots,F_n)}.$$

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution.

$$p(x=v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}}\, e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

For example, suppose the training data contain a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_c be the mean of the values in x associated with class *c*, and let σ_c^2 be the variance of the values in x associated with class *c*. Then, the probability *density* of some value given a class, $P(x = v|c)$, can be computed by plugging v into the equation for a Gaussian distribution parameterized by μ_c and σ_c^2.

As we have already compute the mean value and variance of each training face set and non-face set. We use training set's mean and variance to classify each testing item. We compute the probability of the testing item and face set and non-face set.

posterior(left_eye | face) :
$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

posterior(left_eye | non-face):
$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

Then we compute each feature's probability.
Then the probability that a given document *D* contains all of the words w_i, given a class *C*, is
$$p(D|C) = \prod_i p(w_i|C)$$
Then we compare the posterior(face) and posterior(non-face).

2.2.3 Implementation:
2.2.3.1 Sample Preparation:
We divide the original training sample(face and non-face) into five samples for each. Each sample contains 100 images.

The folder structures are shown in Figure 16:

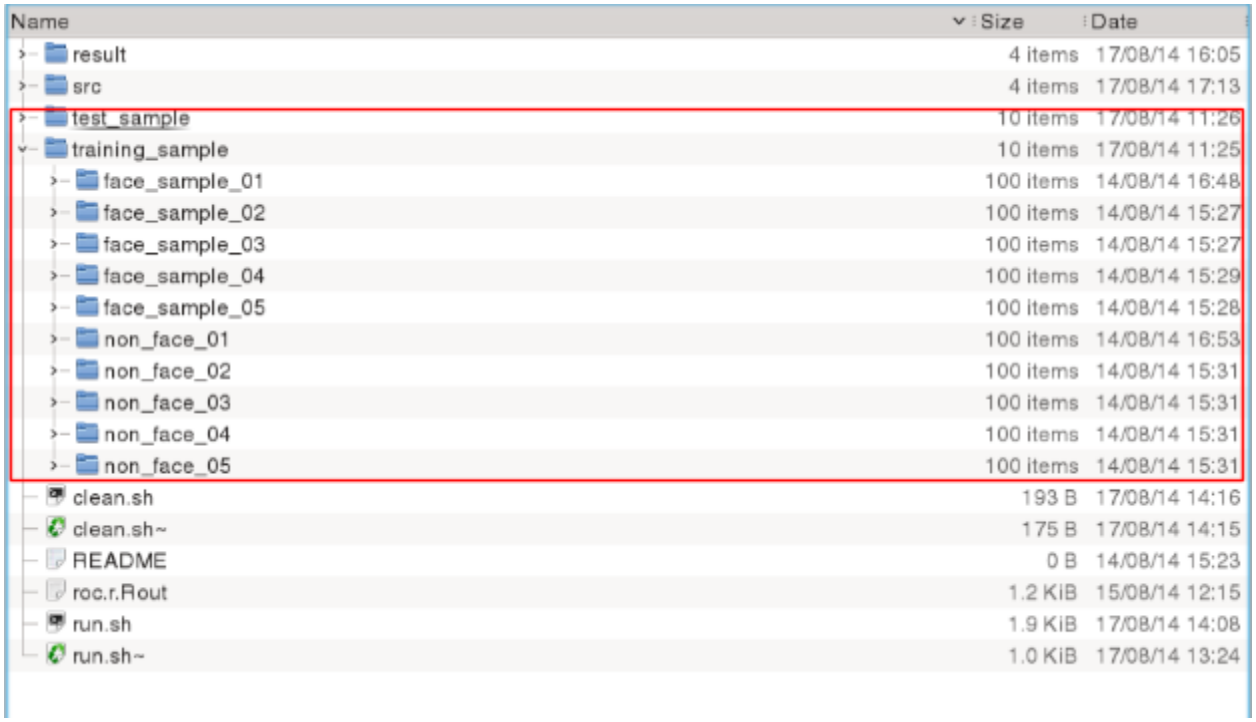


Figure 16

The test sample is under the same structure.

2.2.3.2 Two Classifiers:

In order to discover the importance of feature selection. We made two classifiers that use different features.

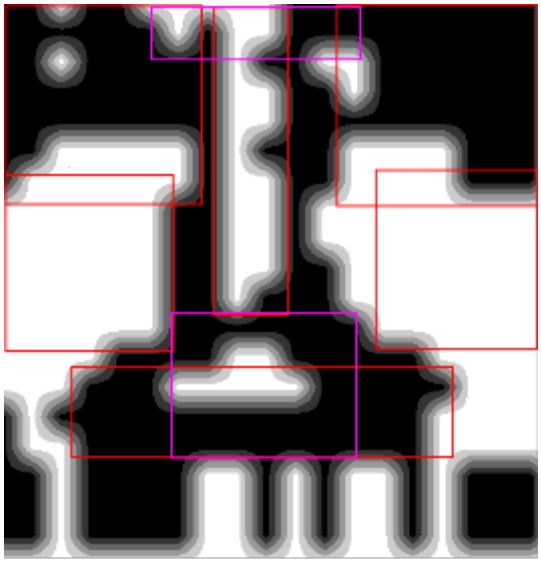


Figure 17: Classifier 1

Figure 18: Classifier 2

In classifier 1, the features are overlapping and large. In contrast, the features in classifier 2 are relatively small and has no overlapping. Although the label of features are same, the result will show differences as we would see in the result analysis section.

2.2.3.3 Procedure

As we divide the original training set into 5 face training sets and 5 non-face training sets. We also divide the test set into the same structure.

In the procedure, we use No.01 face training set and No.01 non-face training set to train the classifier. Then we use the classifier to classify No.01 face testing set and No.01 non-face testing set. After we gain the result, we calculate the TPF/FPF based on it.

Apparently, one set of TPF/FPF data is not enough for us to draw ROC curve.
In order to collect more data, we use each pair of training sets to classify each test set. As we show an example in Figure 19.

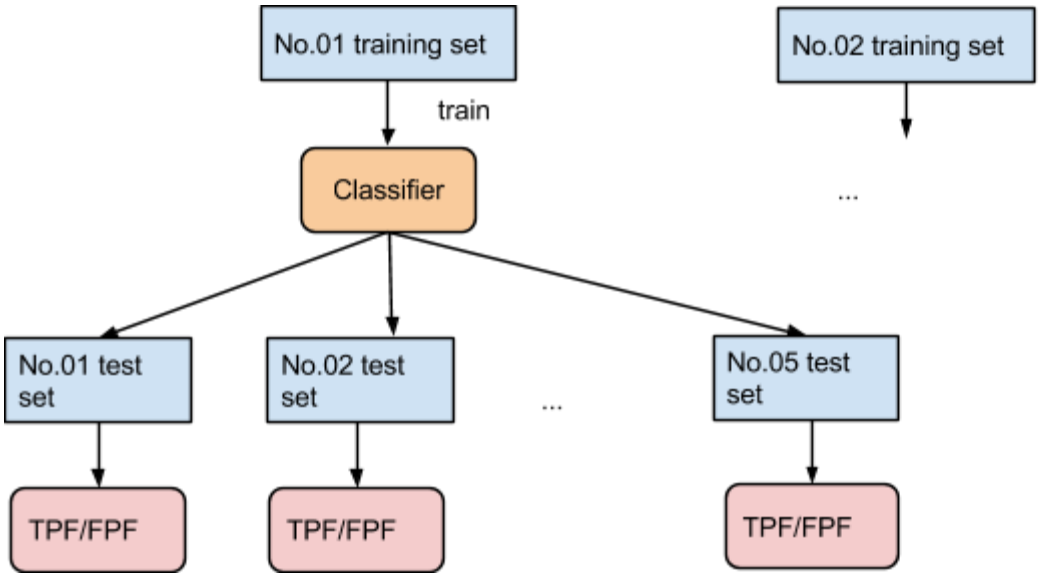


Figure 19

We use 5 pairs of training set to train 5 classifiers, then use each of them classify 5 pairs of test sets. Eventually, we would get 25 sets of TPF/FPF.

2.2.4 Result Analysis

left_eye	forehead	right_eye	nose_bridge	left_cheek	right_cheek	nose	mouth
0.67	0.3	0.8	0.17	0.06	0.1	0.83	1.0
0.65	0.1	0.9	0.22	0.0	0.12	1.0	0.85
0.63	0.0	0.8	0.0	0.03	0.07	0.83	0.85
0.63	0.6	0.69	0.17	0.08	0.05	0.83	0.95

Face-Training result Example

Face-Training result example shows the features we extracted from face images. Each row represents an image. Each column represents a feature vector. There will be 100 rows in each result dataset corresponding to 100 samples we used in each sample set.

After the training process, there will be 5 face-training results and 5 non-face-training results.

left_eye	forehead	right_eye	nose_bridge	left_cheek	right_cheek	nose	mouth
0.6904	0.242	0.6762	0.3692	0.2913	0.1743	0.5517	0.795
0.0268	0.0496	0.0246	0.0483	0.0701	0.0397	0.0658	0.0268

Mean value and variance result Example

Mean value and variance result shows the mean value and variance of each training sample. The first row represents the mean value of each feature. The second row represents the variance of each feature.

After the training process, there will be 5 face-training results and 5 non-face-training results.

result	left_eye	forehead	right_eye	nose_bridge	left_cheek	right_cheek	nose	mouth
1	0.37	0.6	0.78	0.22	0.0	0.07	0.0	0.75
0	0.82	0.4	0.55	0.0	0.83	0.0	0.25	0.25
1	0.59	0.4	0.51	0.0	0.36	0.0	0.58	0.4
0	0.8	0.2	0.35	0.0	0.81	0.0	0.08	0.25

Test sample result Example

Test sample result example shows the result after the test process.

Each row represents a tested image. Result column shows the predicted result. 0 means detected as a non-face, 1 means detected as a face. The other column shows the feature vectors.There will be 100 rows in each result dataset corresponding to 100 samples we used in each sample set.

After the test process, there will be 25 face-test results and 25 non-face-test results.

tp	fp
0	0
1	1
0.62	0.22
0.73	0.22
0.71	0.14
0.63	0.34

TPF/FPF result Example

The TPF/FPF result example shows the result of TPF/FPF we gain from the results sets. True positive fraction(Image is human face, detected as human face). False positive fraction(Image is not human face, detected as human face). The first two rows represent the starting point and end point of ROC curve. There will be 27 rows including starting point and end point as well as 25 results.

This file will be store in ./result/classifier_1/tp_fp.csv.

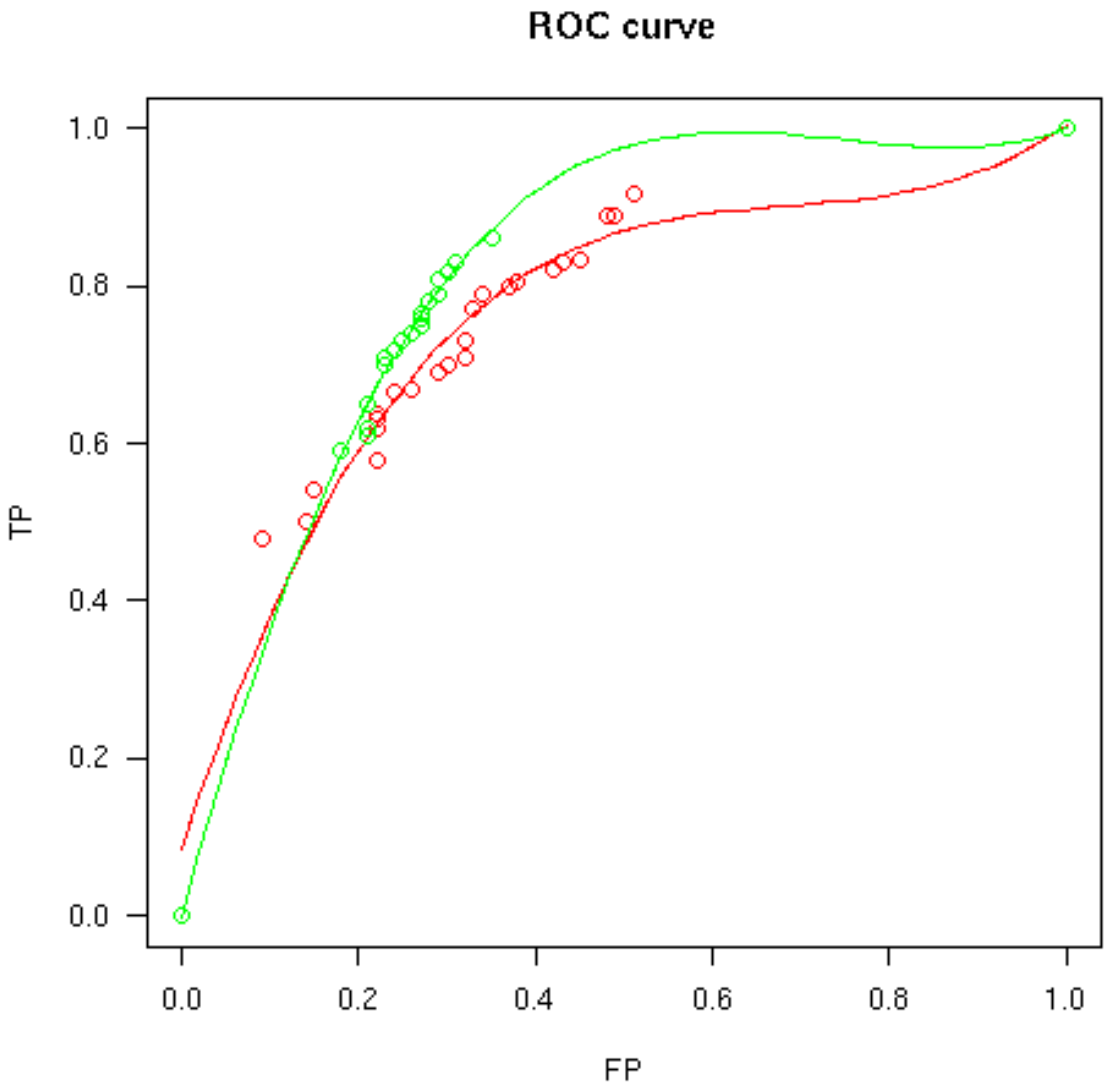


Figure 20

At last, we use R language to draw the ROC curve(Figure 20). As we can see there are 2 curves, each represents a classifier.

The red curve represents the classifier 1.
The curve function is as follow:

$$y = 2.36184444x^3 - 4.84715272x^2 + 3.40435696x + 0.08527465$$

The green curve represents the classifier 2
The curve function is as follow:

$$y = 2.603221111x^3 - 5.814209770x^2 + 4.212181708x - 0.001254369$$

As we can see, the classifier 2 on average produces better results than classifier1. The reason probably because we use more explicit features.
However, as the ROC curve shows, the best prediction in classifier 1 produces 91% in TPF. This prediction is much better than the best prediction in classifier 2, which is 86% in TPF. The reason could be as we select more explicit feature in classifier 2, it could leads to overfitting problem.

Question 3 Data Mining Using Extracted Features

In this question, we are provided with a large number of extracted features from a collection of handwritten numerals ('0'~'9'). 200 patterns per class(for a total of 2,000 patterns) have been digitised in binary images. The total number of features is 649 distributed over 6 files.

What we need to do is as follow:
1. Put the features together in a file.
2. Generate labels for each number.
3. Generate labels for each feature.
4. Make the feature can be used in Weka.
5. Train a C4.5/J48 decision tree classifier using the dataset.
6. Constructed the decision tree and discuss about it.
7. Compare the result of using all feature with using only morphological feature.

3.1 Explain Methods and Implementation:

The first 4 questions are pure engineering problems that related to string feature handle ability.

As for these 4 questions, we use shell script language to sort out the features.
The major tools we used are as follow:

sed
For example:
sed 's/ \+/ /g' mfeat-fac

This could replace multi-whitespace in the input file with one whitespace.

paste
For example:
paste mfeat_fac mfeat_kar > new_file
This example could paste two files column by column into the new_file.

The format for Weka is as follow:
@relation numbers
@attribute att01 numeric
...
@data

After run the run.sh script in Question 3 folder, the sorted file will be generated in the result folder.

./result/
- -mfeat-train.arff
- -mfeat-test.arff
- -mfeat-mor-train.arff
- -mfeat-mor-test.arff

mfeat-train.arff and mfeat-test.arff contain all 649 features.
Each of the file contains 100 records for '0' ~ '9'. So the total records in each file is 1,000.

mfeat-mor-train.arff and mfeat-mor-test.arff contain only morphological features.
Each of the file contains 100 records for '0' ~ '9'. So the total records in each file is 1,000.

3.2 Results Analysis:
In the implementation section, we divide the data into training set and test set. As we use training set to feed the J48 classifier in Weka, the result are as follow:

J48 pruned tree

```
pix48 <= 0
|   pix52 <= 1.524258
|   |   c108 <= 4: 4 (2.0)
|   |   c108 > 4: 1 (93.0/1.0)
|   pix52 > 1.524258
|   |   c185 <= 1052
|   |   |   pix229 <= 1
|   |   |   |   pix91 <= 3: 5 (3.0)
|   |   |   |   pix91 > 3: 2 (3.0)
|   |   |   pix229 > 1
|   |   |   |   c48 <= 8: 1 (2.0)
|   |   |   |   c48 > 8: 4 (98.0/1.0)
|   |   c185 > 1052
|   |   |   f71 <= 0.478731
|   |   |   |   c198 <= 1049
|   |   |   |   |   f11 <= 1.73218
|   |   |   |   |   |   c101 <= 713: 1 (6.0/1.0)
|   |   |   |   |   |   c101 > 713: 7 (97.0/1.0)
|   |   |   |   |   f11 > 1.73218: 3 (13.0)
|   |   |   |   c198 > 1049
|   |   |   |   |   pix201 <= 5
|   |   |   |   |   |   pix49 <= 3
|   |   |   |   |   |   |   zer32 <= 1: 7 (3.0/1.0)
|   |   |   |   |   |   |   zer32 > 1: 2 (96.0/1.0)
|   |   |   |   |   |   pix49 > 3: 3 (4.0)
|   |   |   |   |   pix201 > 5
|   |   |   |   |   |   pix116 <= 5: 3 (65.0)
|   |   |   |   |   |   pix116 > 5: 5 (4.0)
|   |   |   f71 > 0.478731
|   |   |   |   pix116 <= 2
|   |   |   |   |   pix98 <= 0: 3 (16.0)
|   |   |   |   |   pix98 > 0: 5 (2.0)
|   |   |   |   pix116 > 2: 5 (91.0)
pix48 > 0
|   pix50 <= 0: 0 (100.0)
|   pix50 > 0
|   |   c106 <= 4: 6 (99.0/1.0)
|   |   c106 > 4
```

```
|   |   |   pix48 <= 1
|   |   |   |   pix205 <= 2
|   |   |   |   |   pix111 <= 0: 1 (2.0/1.0)
|   |   |   |   |   pix111 > 0: 9 (99.0)
|   |   |   |   pix205 > 2
|   |   |   |   |   pix79 <= 4: 6 (2.0)
|   |   |   |   |   pix79 > 4: 8 (5.0)
|   |   |   pix48 > 1: 8 (95.0)

Number of Leaves  : 24
Size of the tree :      47
```

As we can see from the decision tree. It does not use all 649 features, but it picks 24 features that could already separate 10 numbers.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
100	0	0	0	0	0	0	0	0	0	a = 0
0	100	0	0	0	0	0	0	0	0	b = 1
0	1	98	0	0	0	0	1	0	0	c = 2
0	0	0	98	1	0	0	1	0	0	d = 3
0	0	0	0	99	0	1	0	0	0	e = 4
0	0	0	0	0	100	0	0	0	0	f = 5
0	0	0	0	0	0	100	0	0	0	g = 6
0	1	1	0	0	0	0	98	0	0	h = 7
0	0	0	0	0	0	0	0	100	0	i = 8
0	1	0	0	0	0	0	0	0	99	j = 9

This confusion matrix not only provide correct rate but also shows where the data is misclassified.
For example:
The last row shows the result of number '9', 99% of number '9' can be successfully classified. 1% is misclassified as number '2' for some reason.

Then we could use it to classify the test dataset.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
97	0	0	0	0	2	0	1	0	0	a = 0
0	92	0	3	3	0	0	2	0	0	b = 1
0	2	97	0	0	1	0	0	0	0	c = 2
0	0	2	87	1	7	0	3	0	0	d = 3
0	5	2	2	90	1	0	0	0	0	e = 4
0	0	0	5	2	93	0	0	0	0	f = 5
0	1	0	0	2	0	97	0	0	0	g = 6
0	1	2	4	0	0	0	93	0	0	h = 7
0	0	0	0	0	0	0	0	98	2	i = 8
0	5	0	0	1	2	0	0	0	92	j = 9

This is the result that we use classifier to classify the test dataset.
The accuracy remain above 90% overall. Although number '3' is frequently being classified as number '5'.

Then we use only **morphological features** to train the classifier.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
100	0	0	0	0	0	0	0	0	0	a = 0
0	95	0	0	1	0	1	3	0	0	b = 1
0	1	89	4	0	2	1	3	0	0	c = 2
0	3	4	74	11	6	0	2	0	0	d = 3
0	2	1	6	86	0	1	4	0	0	e = 4
0	0	19	10	5	66	0	0	0	0	f = 5
0	0	0	0	0	0	100	0	0	0	g = 6
0	1	1	0	2	0	0	96	0	0	h = 7
0	0	0	0	0	0	0	0	100	0	i = 8
0	0	0	0	0	1	99	0	0	0	j = 9

As we can observe from the confusion matrix, some of the numbers could be successfully classified. For example: number '1', '6' and '8'. Some of the numbers can mostly be classified, but for number '5' , it always misclassified as '2' and '3'. There is one particular case, number '9' could not be classified by morphological features, as it will be classified as '6'.

If we use this classifier to classify the test sample. We would gain a similar result.

=== Confusion Matrix ===

```
  a  b  c  d  e  f  g  h  i  j   <-- classified as
 97  0  1  0  0  2  0  0  0  0 |  a = 0
  0 93  1  0  3  0  0  3  0  0 |  b = 1
  0  0 70  4  5 16  1  4  0  0 |  c = 2
 0  0  8 53 21  8  0 10  0  0 |  d = 3
  0  5  3  9 65  4  0 14  0  0 |  e = 4
 0  0 15 15  2 66  0  2  0  0 |  f = 5
   0  1  0  1  0  0 98  0  0  0 |  g = 6
  0  3  6  3 10  0  0 78  0  0 |  h = 7
    0  0  0  0  0  0  2  0 98  0 |  i = 8
    0  1  3  0  0  1 95  0  0  0 |  j = 9
```

3.3 Conclusion:

Although by using morphological features, the J48(decision tree) classifier could successfully classify most of the hand written numbers, but for particular case such as number '9', the shape information does not sufficiently provide a good result. If we use all 649 features, although the classifier does not necessarily use all the features, it only select a set of features which could also provide a reasonably good result.

References:

1.OTSU, N. 1979. A threshold selection method from gray-level histogram. IEEE Transactions on Systems, Man and Cybemetics 9, 1, 62–66.